Matrix Profile X: VALMOD - Scalable Discovery of Variable-Length Motifs in Data Series

Michele Linardi
LIPADE, Paris Descartes University
michele.linardi@parisdescartes.fr

Yan Zhu
UC Riverside
yzhu015@ucr.edu

Themis Palpanas
LIPADE, Paris Descartes University
themis@mi.parisdescartes.fr

Eamonn Keogh
UC Riverside
eamonn@cs.ucr.edu

ABSTRACT

In the last fifteen years, data series motif discovery has emerged as one of the most useful primitives for data series mining, with applications to many domains, including robotics, entomology, seismology, medicine, and climatology. Nevertheless, the state-of-the-art motif discovery tools still require the user to provide the motif length. Yet, in at least some cases, the choice of motif length is critical and unforgiving. Unfortunately, the obvious brute-force solution, which tests all lengths within a given range, is computationally untenable. In this work, we introduce VALMOD, an exact and scalable motif discovery algorithm that efficiently finds all motifs in a given range of lengths. We evaluate our approach with five diverse real datasets, and demonstrate that it is up to 20 times faster than the state-of-the-art. Our results also show that removing the unrealistic assumption that the user knows the correct length, can often produce more intuitive and actionable results, which could have been missed otherwise.

KEYWORDS

Data Series; Time Series; Motif Discovery; Variable Length; Data Mining

ACM Reference Format:
Michele Linardi, Yan Zhu, Themis Palpanas, and Eamonn Keogh. 2018. Matrix Profile X: VALMOD - Scalable Discovery of Variable-Length Motifs in Data Series. In *SIGMOD/PODS '18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3183713.3183744

1 INTRODUCTION

Data series[1] have gathered the attention of the data management community for more than two decades [1–3, 6, 13–15, 17, 18, 37, 38, 47, 50, 57, 59]. They are now one of the most common types of data, present in virtually every scientific and social domain [16, 26, 35, 39].

Over the last decade, data series motif discovery has emerged as perhaps the most used primitive for data series data mining, and it has many applications to a wide variety of domains [48, 51], including classification, clustering, and rule discovery. More recently, there has been substantial progress on the scalability of motif discovery, and now massive datasets can be routinely searched on conventional hardware [48].

Another critical improvement in motif discovery, is the reduction in the number of parameters requiring specification. The first motif discovery algorithm, PROJECTION [5], required the user to set seven parameters, and it still only produces answers that are approximately correct. Researchers have *"chipped"* away at this over the years [31, 42], and the current state-of-the-art algorithms only require the user to set a single parameter, which is the desired length of the motifs. Paradoxically, the ease with which we can now perform motif discovery has revealed that even this single burden on the user's experience or intuition may be too great.

For example, AspenTech, a company that makes software for optimizing the manufacturing process for the oil and gas industry, has begun to use motif discovery in their products both as a standalone service and also as part of a precursor search tool. They recently noted that, *"our lighthouse (early adopter) customers love motif discovery, and they feel it adds great value [...] but they are frustrated by the finicky setting of the motif length."* [33]. This issue, of being restricted to specify length as an input parameter, has also been noted in other domains that use motif discovery, such as cardiology [45] and speech therapy [46], as well as in related problems, such as data series indexing [22].

The obvious solution to this issue is to make the algorithms search over all lengths in a given range and rank the various length motifs discovered.

Nevertheless, this strategy poses two challenges. First, how we can rank motifs of different lengths. Second, and most important, how we can search over this much larger solution space in an efficient way, in order to identify the motifs.

In this work, we address both problems. The former lends itself to a simple solution, which has not been reported in the literature. The latter requires new techniques that significantly extend the state-of-the-art algorithms, including the introduction of a novel lower bounding method, which makes efficiently searching a large space of potential solutions possible.

[1] If the dimension that imposes the ordering of the series is time, then we talk about *time series*. However, a series can also be defined through other measures (e.g., angle in radial profiles in astronomy, mass in mass spectroscopy, position in genome sequences, etc.). We use the terms *time series*, *data series*, and *sequence* interchangeably.

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3183744

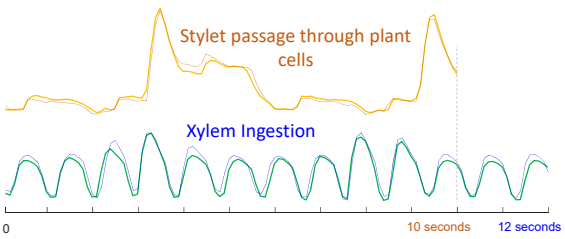

Figure 1: An existence proof of semantically different motifs, of slightly different lengths, extracted from a single dataset.

Note that even if the user has good knowledge of the data domain, in many circumstances, searching with one single motif length is not enough, because the data can contain motifs of various lengths. For example, Figure 1 previews the 10-second and 12-second motifs discovered in the Electrical Penetration Graph (EPG) of an insect called Asian citrus psyllid. The two motif pairs are radically different, reflecting two different types of insect activities. In order to capture all useful activity information within the data, a fast search of motifs over all lengths is necessary.

The contributions of our work are the following.

- We define the problem of variable-length motif discovery, which significantly extends the usability of the motif discovery operation.
- We propose an exact algorithm, called Variable Length Motif Discovery (VALMOD): Given a data series T, VALMOD finds the subsequence pair with the smallest Euclidean distance of each length in the input range $[\ell_{min}, \ell_{max}]$. VALMOD is based on a novel lower bounding technique, which is specifically designed for the motif discovery problem.
- We evaluate our approach using five diverse real datasets, and demonstrate the scalability of our approach. The results show that VALMOD is up to 20x faster than the state-of-the-art-techniques. Furthermore, we present case studies with datasets from entomology and seismology, which demonstrate the usefulness of our approach.

2 PROBLEM DEFINITION

We begin by defining the data type of interest, data series:

Definition 2.1 (Data series). A data series $T \in \mathbb{R}^n$ is a sequence of real-valued numbers $t_i \in \mathbb{R}$ $[t_1, t_2, ..., t_n]$, where n is the length of T.

We are typically not interested in the global properties of a data series, but in the local regions known as subsequences:

Definition 2.2 (Subsequence). A subsequence $T_{i,\ell} \in \mathbb{R}^\ell$ of a data series T is a continuous subset of the values from T of length ℓ starting from position i. Formally, $T_{i,\ell} = [t_i, t_{i+1}, ..., t_{i+\ell-1}]$.

The particular local property we are interested in this work is data series motifs. A data series motif pair is the pair of the most similar subsequences of a given length, ℓ, of a data series:

Definition 2.3 (Data series motif pair). $T_{a,\ell}$ and $T_{b,\ell}$ is a motif pair iff $dist(T_{a,\ell}, T_{b,\ell}) \leq dist(T_{i,\ell}, T_{j,\ell})$ $\forall i, j \in [1, 2, ..., n - \ell + 1]$, where $a \neq b$ and $i \neq j$, and dist is a function that computes the z-normalized Euclidean distance between the input subsequences [5, 31, 46, 48, 51].

Note, that if we remove the motif pair from the dataset, the pair with the second smallest distance will become the new motif pair. In this way, we can produce a ranked list of subsequence pairs, which we call motif pairs of length ℓ.

We store the distance between a subsequence of a data series with all the other subsequences from the same data series in an ordered array called a distance profile.

Definition 2.4 (Distance profile). A distance profile $D \in \mathbb{R}^{(n-\ell+1)}$ of a data series T regarding subsequence $T_{i,\ell}$ is a vector that stores $dist(T_{i,\ell}, T_{j,\ell})$, $\forall j \in [1, 2, ..., n - \ell + 1]$, where $i \neq j$.

One of the most efficient ways to locate the exact data series motif is to compute the matrix profile [53, 56], which can be obtained by evaluating the minimum value of every distance profile in the time series.

Definition 2.5 (Matrix profile). A matrix profile $MP \in \mathbb{R}^{(n-\ell+1)}$ of a data series T is a meta data series that stores the z-normalized Euclidean distance between each subsequence and its nearest neighbor, where n is the length of T and ℓ is the given subsequence length. The data series motif can be found by locating the two lowest values in MP.

To avoid trivial matches [4], in which a pattern is matched to itself or a pattern that largely overlaps with itself, the matrix profile incorporates an "exclusion-zone" concept, which is a region before and after the location of a given query that should be ignored. The exclusion zone is heuristically set to $\ell/2$. The recently introduced STOMP algorithm [56] offers a solution to compute the matrix profile MP in $O(n^2)$ time. This may seem untenable for data series mining, but several factors mitigate this concern. First, note that the time complexity is independent of ℓ, the length of the subsequences. Secondly, the matrix profile can be computed with an anytime algorithm, and in most domains, in just $O(nc)$ steps the algorithm converges to what would be the final solution [53] (c is a small constant). Finally, the matrix profile can be computed with GPUs, cloud computing, and other HPC environments that make scaling to at least tens of millions of data points trivial [56].

We can now formally define the problems we solve.

Problem 1 (Variable-Length Motif Pair Discovery). *Given a data series T and a subsequence length-range $[\ell_{min}, ..., \ell_{max}]$, we want to find the data series motif pairs of all lengths in $[\ell_{min}, ..., \ell_{max}]$, occurring in T.*

One naive solution to this problem is to repeatedly run the state-of-the art motif discovery algorithms for every length in the range. However, note that the size of this range can be as large as $O(n)$, which makes the naive solution infeasible for even middle-size data series. We aim at reducing this $O(n)$ factor to a small value.

Note that the motif pair discovery problem has been extensively studied in this last decade [19, 27–29, 31, 53, 56], the reason being that if we want to find a collection of recurrent subsequences in T, the most computationally expensive operation consists of identifying the motif pairs [56], namely, solving Problem 1. Extending

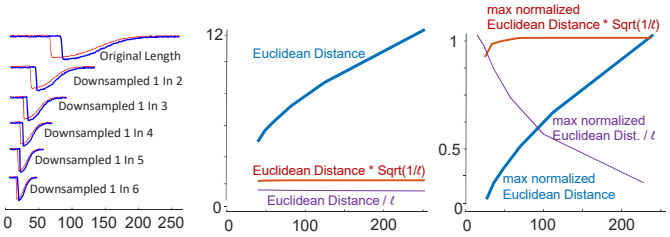

Figure 2: (*left*) Two series from the TRACE dataset, as proxies for time series signatures at various speeds. (*center*) The classic Euclidean distance is clearly not length invariant. (*right*) After divide-by-max normalizing, it is clear that the length-normalized Euclidean distance has a strong bias toward the longest pattern.

motif pairs to sets incurs a negligible additional cost (as we also show in our study).

Given a motif pair $\{T_{\alpha,\ell}, T_{\beta,\ell}\}$, the data series motif set S_r^ℓ, with radius $r \in \mathbb{R}$, is the set of subsequences of length ℓ, which are at distance at most r from either $T_{\alpha,\ell}$, or $T_{\beta,\ell}$. More formally:

Definition 2.6 (Data series motif set). Let $\{T_{\alpha,\ell}, T_{\beta,\ell}\}$ a motif pair of length ℓ of data series T. The motif set S_r^ℓ is defined as: $S_r^\ell = \{T_{i,\ell} | dist(T_{i,\ell}, T_{\alpha,\ell}) < r \vee dist(T_{i,\ell}, T_{\beta,\ell}) < r\}$.

The cardinality of S_r^ℓ, $|S_r^\ell|$, is called the frequency of the motif set.

Intuitively, we can build a motif set starting from a motif pair. Then, we iteratively add in the motif set all subsequences within radius r. We use the above definition to solve the following problem (optionally including a constraint on the minimum frequency for motif sets in the final answer).

Problem 2 (Variable-Length Motif Sets Discovery). *Given a data series T and a length range $[\ell_{min}, \ldots, \ell_{max}]$, we want to find the set $S^* = \{S_r^\ell | S_r^\ell$ is a motif set, $\ell_{min} \leq \ell \leq \ell_{max}\}$. In addition, we require that if $S_r^\ell, S_{r'}^{\ell'} \in S^* \Rightarrow S_r^\ell \cap S_{r'}^{\ell'} = \emptyset$.*

Thus, the variable-length motif sets discovery problem results in a set, S^*, of motif sets. The constraint at the end of the problem definition restricts each subsequence to be included in at most one motif set. Note that in practice we may not be interested in all the motif sets, but only in those with the K smallest distances, leading to a top-K version of the problem. In our work, we provide a solution for the top-K problem (though, setting K to a very large value will produce all results).

3 COMPARING MOTIFS OF DIFFERENT LENGTHS

Before introducing our solutions to the problems outlined above, we first discuss the issue of comparing motifs of different lengths. This becomes relevant when we want to rank motifs of different lengths (within the given range), which is useful in order to identify the most prominent motifs, irrespective of their length. In this section, we propose a length-normalized distance measure that the VALMOD algorithm uses in order to produce such rankings.

The increased expressiveness of VALMOD offers a challenge. Since we can *discover* motifs of different lengths, we also need to be able to *rank* motifs of different lengths. A similar problem occurs in string processing, and a common solution is to replace the edit-distance by the length-normalized edit-distance, which is the classic distance measure divided by the length of the strings in question [24]. This correction would find the pair {concatenation, concameration} *more* similar than {cat, cot}, matching our intuition, since only 15% of the characters are different in the former pair, as opposed to 33% in the latter.

Researchers have suggested this length-normalized correction for time series, but as we will show, the correction factor is incorrect. To illustrate this, consider the following thought experiment. Imagine that some process in the system we are monitoring occasionally "injects" a pattern into the time series. As a concrete example, washing machines typically have a prototypic signature (as exhibited in the TRACE dataset [40]), but the signatures express themselves more slowly on a cold day, when it takes longer to heat the cooler water supplied from the city [9]. We would like all equal length instances of the signature to have approximately the same distance. As a consequence, we factorize the Euclidean distance by the following quantity: $sqrt(1/\ell)$, where ℓ is the length of the sequences. This aims to favor longer and similar sequences in the ranking process of matches that have different lengths.

In Figure 2(*left*) we show two examples from the TRACE dataset [40], which will act as proxies for a variable length signature. We produced the variable lengths by down sampling. In Figure 2(*center*), we show the distances between the patterns as their length changes. With no correction, the Euclidean distance is obviously biased to the shortest length. The length-normalized Euclidean distance looks "flatter" and suggests itself as the proper correction. However, its variation over the sequence length change is not visible due to the small scale. In Figure 2(*right*), we show all of the measures after dividing them by their largest value. Now we can see that the length-normalized Euclidean distance has a strong bias toward the longest pattern. In contrast to the other two approaches, the $sqrt(1/length)$ correction factor provides a near perfect invariant distance over the entire range of values.

4 PROPOSED APPROACH

Our algorithm, VALMOD (Variable Length Motif Discovery), starts by computing the *matrix profile* on the smallest subsequence length, namely ℓ_{min}, within a specified range $[\ell_{min}, \ell_{max}]$. The key idea of our approach is to minimize the work that needs to be done for succeeding subsequence lengths ($\ell_{min}+1$, $\ell_{min}+2$, ..., ℓ_{max}). In Figure 3, it can be observed that the motif of length 8 ($T_{33,8} - T_{97,8}$) have the same offsets as the motif of length 9 ($T_{33,9} - T_{97,9}$). Can we exploit this property to accelerate our computation?

It seems that if the nearest neighbor of $T_{i,\ell_{min}}$ is $T_{j,\ell_{min}}$, then probably the nearest neighbor of $T_{i,\ell_{min}+1}$ is $T_{j,\ell_{min}+1}$. For example, as shown in Figure 3(bottom), if we sort the distance profiles of $T_{33,8}$ and $T_{33,9}$ in ascending order, we can find that the nearest neighbor of $T_{33,8}$ is $T_{97,8}$, and the nearest neighbor of $T_{33,9}$ is $T_{97,9}$.

One can imagine that if the location of the nearest neighbor of $T_{i,\ell}$ ($i = 1, 2, ..., n-m+1$) remains the same as we increase ℓ, then we could obtain the matrix profile of length $\ell + k$ in $O(n)$ time

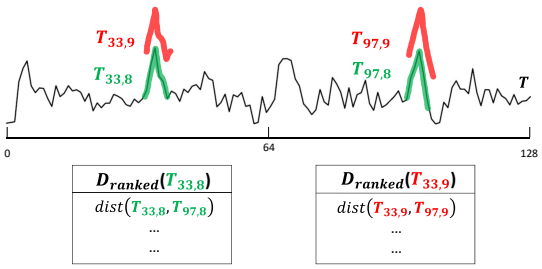

Figure 3: (*top*) The top motifs of length 9 in an example data series have the same offsets as the top motifs of length 8. (*bottom*) The sorted distance profiles of $T_{33,8}$ and $T_{33,9}$.

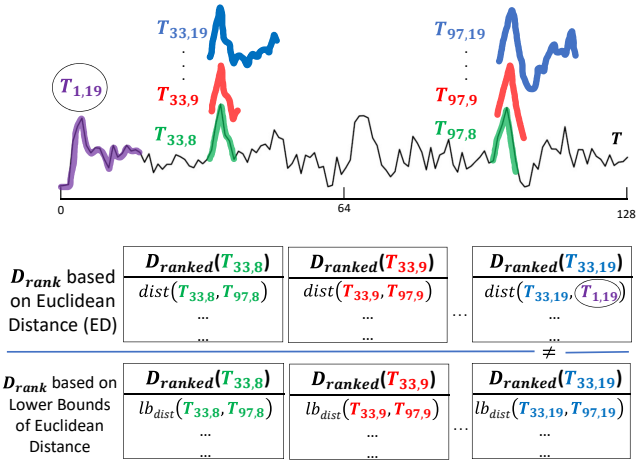

Figure 4: (*top distance profiles*) Ranking by true distances leads to changes in the order of the pairs. (*bottom distance profiles*) Ranking by lower bound distances maintains the same order of pairs over increasing lengths.

($k = 1, 2, \ldots$). However, this is not always true. The location of the nearest neighbor of $T_{i,\ell}$ may not change as we slightly increase ℓ, if there is a substantial margin between the first and second entries of $D_{ranked}(T_{i,\ell})$. But, as ℓ gets larger, the nearest neighbor of $T_{i,\ell}$ is likely to change. For example, as shown in Figure 4, when the subsequence length grows to 19, the nearest neighbor of $T_{33,19}$ is no longer $T_{97,19}$, but $T_{1,19}$. We observe that the ranking of the distance profile values may change, even when the data is relatively smooth. When the data is noisy and skewed, this ranking can change even more often. Is there any other rank-preserving measure that we can exploit to accelerate the computation?

The answer is yes. Instead of sorting the entries of the distance profile, we create and sort a new vector, called the *lower bound distance profile*. Figure 4(bottom) previews the rank-preserving property of the lower bound distance profile. As we will describe later, once we know the distance between $T_{i,\ell}$ and $T_{j,\ell}$, we can evaluate a lower bound distance between $T_{i,\ell+k}$ and $T_{j,\ell+k}$, $\forall k \in [1,2,3,\ldots]$. The rank-preserving property of the lower bound distance profile can help us prune a large number of unnecessary computations as we increase the subsequence length.

4.1 The Lower Bound Distance Profile

Before introducing the lower bound distance profile, let us first investigate its basic element: the lower bounding Euclidean distance.

Assume that we already know the z-normalized Euclidean distance $d^{\ell}_{i,j}$ between two subsequences of length ℓ: $T_{i,\ell}$ and $T_{j,\ell}$, and we are now estimating the distance between two longer subsequences of length $\ell + k$: $T_{i,\ell+k}$ and $T_{j,\ell+k}$. Our problem can be stated as follows: given $T_{i,\ell}$, $T_{j,\ell}$ and $T_{j,\ell+k}$ (but not the last k values of $T_{i,\ell+k}$), is it possible to provide a lower bound function $LB(d^{\ell+k}_{i,j})$, such that $LB(d^{\ell+k}_{i,j}) \leq d^{\ell+k}_{i,j}$? This problem is visualized in Figure 5 .

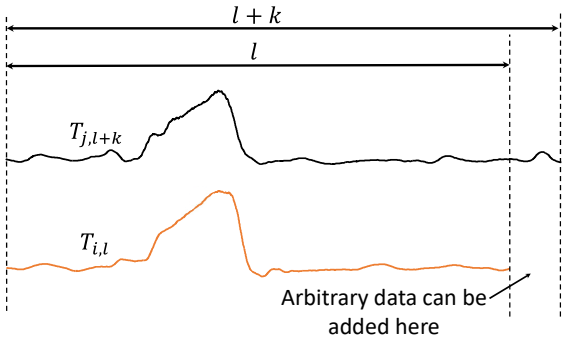

Figure 5: Increasing the subsequence length from ℓ to $\ell + k$.

One may assume that we can simply set $LB(d^{\ell+k}_{i,j}) = d^{\ell}_{i,j}$ by assuming that the last k values of $T_{i,\ell+k}$ are the same as the last k values of $T_{j,\ell+k}$. However, this is not an answer to our problem, as we need to evaluate z-normalized Euclidean distances, which are not simple Euclidean distances. The mean and standard deviation of a subsequence can change as we increase its length, so we need to re-normalize both $T_{i,\ell+k}$ and $T_{j,\ell+k}$. Assume that the mean and standard deviation of $T_{x,y}$ are $\mu_{x,y}$ and $\sigma_{x,y}$, respectively (i.e. $T_{j,\ell+k}$ corresponds to $\mu_{j,\ell+k}$ and $\sigma_{j,\ell+k}$). Since we do not know the last k values of $T_{i,\ell+k}$, both $\mu_{i,\ell+k}$ and $\sigma_{i,\ell+k}$ are unknown and can thus be regarded as variables, and we have the following [55]:

$$d^{\ell+k}_{i,j} \geq \min_{\mu_{i,\ell+k},\sigma_{i,\ell+k}} \sqrt{\sum_{p=1}^{\ell} \left(\frac{t_{i+p-1} - \mu_{i,\ell+k}}{\sigma_{i,\ell+k}} - \frac{t_{j+p-1} - \mu_{j,\ell+k}}{\sigma_{j,\ell+k}}\right)^2}$$

$$= \min_{\mu_{i,\ell+k},\sigma_{i,\ell+k}} \frac{\sigma_{j,\ell}}{\sigma_{j,\ell+k}} \sqrt{\sum_{p=1}^{\ell} \left(\frac{t_{i+p-1} - \mu_{i,\ell+k}}{\frac{\sigma_{i,\ell+k}\sigma_{j,\ell}}{\sigma_{j,\ell+k}}} - \frac{t_{j+p-1} - \mu_{j,\ell+k}}{\sigma_{j,\ell}}\right)^2}$$

$$= \min_{\mu',\sigma'} \frac{\sigma_{j,\ell}}{\sigma_{j,\ell+k}} \sqrt{\sum_{p=1}^{\ell} \left(\frac{t_{i+p-1} - \mu'}{\sigma'} - \frac{t_{j+p-1} - \mu_{j,\ell}}{\sigma_{j,\ell}}\right)^2} \quad (1)$$

Clearly, the minimum value shown in Eq. 1 can be set as $LB(d^{\ell+k}_{i,j})$. We can obtain $LB(d^{\ell+k}_{i,j})$ by solving $\frac{\partial LB(d^{\ell+k}_{i,j})}{\partial \mu'} = 0$ and $\frac{\partial LB(d^{\ell+k}_{i,j})}{\partial \sigma'} = 0$:

$$LB(d_{i,j}^{\ell+k}) = \begin{cases} \sqrt{\ell}\frac{\sigma_{j,\ell}}{\sigma_{j,\ell+k}} & \text{if } q_{i,j} \leq 0 \\ \sqrt{\ell(1-q_{i,j}^2)}\frac{\sigma_{j,\ell}}{\sigma_{j,\ell+k}} & \text{otherwise} \end{cases} \quad (2)$$

where $q_{i,j} = \frac{\sum_{p=1}^{\ell}\frac{(t_{j+p-1}t_{i+p-1})}{\ell} - \mu_{i,\ell}\mu_{j,\ell}}{\sigma_{i,\ell}\sigma_{j,\ell}}$.

$LB(d_{i,j}^{\ell+k})$ yields the minimum possible z-normalized Euclidean distance between $T_{i,\ell+k}$ and $T_{j,\ell+k}$, given $T_{i,\ell}$, $T_{j,\ell}$ and $T_{j,\ell+k}$ (but not the last k values of $T_{i,\ell+k}$). Now that we have obtained the lower bound Euclidean distance between two subsequences, we are able to introduce the lower bound distance profile.

Using Eq. 2, we can evaluate the lower bound Euclidean distance between $T_{j,\ell+k}$ and every subsequence of length $\ell + k$ in T. By putting the results in a vector, we obtain the lower bound distance profile $LB(D_j^{\ell+k})$ corresponding to subsequence $T_{j,\ell+k}$: $LB(D_j^{\ell+k})$ = $LB(d_{1,j}^{\ell+k})$, $LB(d_{2,j}^{\ell+k})$, ...,$LB(d_{n-\ell-k+1,j}^{\ell+k})$. If we sort the components of $LB(D_j^{\ell+k})$ in an ascending order, we can obtain the ranked lower bound distance profile: $LB_{ranked}(D_j^{\ell+k}) = LB(d_{r_1,j}^{\ell+k}), LB(d_{r_2,j}^{\ell+k})$, ..., $LB(d_{r_{n-\ell-k+1},j}^{\ell+k})$, where $LB(d_{r_1,j}^{\ell+k}) \leq LB(d_{r_2,j}^{\ell+k}) \leq \ldots \leq LB(d_{r_{n-\ell-k+1},j}^{\ell+k})$.

We would like to utilize this ranked lower bound distance profile to accelerate our computation. Assume that we have a best-so-far pair of motifs with a distance $dist_{BSF}$. If we examine the p^{th} element in the ranked lower bound distance profile and find that $LB(d_{r_p,j}^{\ell+k}) > dist_{BSF}$, then we do not need to calculate the exact distance for $d_{r_p,j}^{\ell+k}, d_{r_{p+1},j}^{\ell+k}, \ldots, d_{r_{n-\ell-k+1},j}^{\ell+k}$ anymore, as they cannot be smaller than $dist_{BSF}$. Based on this observation, our strategy is as follows. We set a small, fixed value for p. Then, for every j, we evaluate whether $LB(d_{r_p,j}^{\ell+k}) > dist_{BSF}$ is true: if it is, we only calculate $d_{r_1,j}^{\ell+k}, d_{r_2,j}^{\ell+k}, \ldots, d_{r_{p-1},j}^{\ell+k}$. If it is not, we compute all the elements of $D_j^{\ell+k}$. We update $dist_{BSF}$ whenever a smaller distance value is observed. In the best case, we just need to calculate $O(np)$ exact distance values to obtain the motif of length $l + k$. Note that the order of the ranked lower bound distance profile is preserved for every k. That is to say, if $LB(d_{a,j}^{\ell+k}) \leq LB(d_{b,j}^{\ell+k})$, then $LB(d_{a,j}^{\ell+k+1}) \leq LB(d_{b,j}^{\ell+k+1})$. This is because the only component in Eq. 2 related to k is $\sigma_{j,\ell+k}$. When we increase k by 1, we are just performing a linear transformation for the lower bound distance: $LB(d_{i,j}^{\ell+k+1}) = LB(d_{i,j}^{\ell+k})\sigma_{j,\ell+k}/\sigma_{j,\ell+k+1}$. Therefore, we have $LB(d_{r_p,j}^{\ell+k+1}) = LB(d_{r_p,j}^{\ell+k})\sigma_{j,\ell+k}/\sigma_{j,\ell+k+1}$, and the ranking is preserved for every k.

4.2 The VALMOD Algorithm

We are now able to formally describe the VALMOD algorithm. The pseudocode for VALMOD is shown in Algorithm 1. With the call of *ComputeMatrixProfile()* in line 5, we build the matrix profile corresponding to ℓ_{min}, and in the meantime store the smallest p values of each distance profile in the memory. Note that the matrix profile is accommodated in the vector MP, which is coupled with the matrix profile index, IP, which is a structure containing the offsets of the nearest neighbor subsequences. We can easily find the motif corresponding to ℓ_{min} as the minimum value of MP.

Algorithm 1: *VALMOD*

```
   Input: DataSeries T, int ℓmin int ℓmax, int p
   Output: VALMP
1  int nDP ← |T| − ℓmin + 1 ;
2  VALMP ← new VALMP(nDP);
3  VALMP.MP = {⊥,...,⊥};
4  MaxHeap[] listDP , double [] MP, int [] IP ;
5  listDP, MP , IP ← ComputeMatrixProfile(T, ℓmin, p); // listDP contains p
       entries of each distance profile
6  VALMP ← updateVALMP(VALMP,MP,IP,nDP) ;
7  for i ← ℓmin + 1 to ℓmax do
8      nDP ← |T| − i + 1 ;
       // compute SubMP and update listDP for the length i
9      bool bBestM, double [] SubMP, IP ← ComputeSubMP(T,nDP,listDP,i,p);
10     if bBestM then
           // SubMP surely contains the motif, update VALMP with it
11         updateVALMP(VALMP,SubMP,IP,nDP);
12     else
13         listDP,MP,IP ← ComputeMatrixProfile(T,i,p);
           // SubMP might not contain the motif, update VALMP computing MP
14         updateVALMP(VALMP,MP,IP,nDP);
15     end
16 end
```

Algorithm 2: *updateVALMP*

```
   Input: VALMP, double [] MPnew, int [] IP, nDP, ℓ
   Output: VALMP
1  for i ← 1 to nDP do
       // length normalize the Euclidean distance
2      double lNormDist ← MPnew[i] ∗ √(1/ℓ);
       // if the distance at offset i of VALMP, surely comptued with previous
          lengths, is larger than the actual, update it
3      if (VALMP.distances[i] > lNormDist or VALMP.MP[i] == ⊥) then
4          VALMP.distances[i] ← MPnew[i];
5          VALMP.normDistances[i] ← lNormDist;
6          VALMP.lengths[i] ← ℓ;
7          VALMP.indices[i] ← IP[i];
8      end
9  end
```

Then in lines 7-16, we iteratively look for the motif of every length within ℓ_{min}+1 and ℓ_{max}. The *ComputeSubMP* function in line 9 attemps to find the motif of length i by only evaluating a subset of the matrix profile corresponding to subsequence length i. Note that this strategy, which is based on the lower bounding technique introduced in section 4.1, might not be able to capture the global minimum value within the matrix profile. In case that happens (which is rare), the boolean flag *bBestM* is set to false, and we compute the whole matrix profile with the *computeMatrixPorilfe* procedure in line 13.

The final output of *VALMOD* is a vector, which is called *VALMP* (*variable length matrix profile*) in the pseudo-code. If we were interested in only one fixed subsequence length, *VALMP* would be the matrix profile normalized by the square root of the subsequence length. If we are processing various subsequence lengths, then as we increase the subsequence length, we update *VALMP* when a smaller length-normalized Euclidean distance is observed.

Algorithm 2 shows the routine to update the *VALMP* structure. The final *VALMP* consists of four parts. The i^{th} entry of the *normDistances* vector stores the smallest length-normalized Euclidean distance values between the i^{th} subsequence and its nearest neighbor, while the i^{th} place of vector *distances* stores their straight Euclidean distance. The location of each subsequence's nearest

Algorithm 3: *ComputeMatrixProfile*

```
   Input: DataSeries T, int ℓ, int p
   Output: MP, listDP
1  int nDP ← |T|-ℓ+1;
2  double [] MP ← double [nDP];
3  int [] IP ← int [nDP];
4  MaxHeap[] listDP= new MaxHeap(p)[nDP];
   // compute the dot product vector QT for the first distance profile
5  double [] QT ← SlidingDotProduct(T_{1,ℓ}, T);
   // compute sum and squared sum of the first subsequence of length ℓ
6  s ← sum(T_{1,ℓ}); ss ← squaredSum(T_{1,ℓ});
   // comptue the first distance profile with distance formula (Eq.3) and store
      the minimum distance in MP and the offset of the nearest neighbor in IP
7  D(T_{i,ℓ}) ← CalcDistProfile(QT,T_{i,ℓ}, T, s, ss);
8  MP[1], IP[1] ← min(D(T_{i,ℓ}));
   // iterate over the subsequences of T
9  for i ← 2 to nDP do
       // update the dot product vector QT for the i^th subsequence
10     for j ← nDP down to 2 do
11         QT[j]←QT[j − 1] − T[j − 1] × T[i − 1] + T[j + ℓ − 1] × T[i + ℓ − 1] ;
12     end
       // update sum and squared sum of the i^th subsequence
13     s ← s − T[i − 1] + T[ℓ + i − 2];
14     ss ← ss − T[i − 1]^2 + T[ℓ + i − 2]^2;
15     D(T_{i,ℓ}) ← CalcDistProfile(QT,T_{i,ℓ}, T, s, ss);
16     MP[i], IP[i] ← min(D(T_{i,ℓ}));
       // Store in listDP[i] the p entries e with smallest lower bounding
          distance
17     int c ← 0;
18     for each entry e in D(T_{i,ℓ}) do
           // Compute the lower bound for the length ℓ + 1
19         e.LB ← compLB(ℓ, ℓ + 1, QT[c], e.s1, e.s2, e.ss1, e.ss2);
           // save the entry only if is smaller than the max lb so far or if
              listDP[i] contains less than p elements
20         if e.LB < max(listDP[i]) or |listDP[i]| < p then
21             insert(listDP[i], e);
22         end
23         c← c + 1;
24     end
25 end
```

neighbor is stored in the vector *indices*. The structure *lengths* contains the length of the i^{th} subsequences pair.

In the next two subsections, we detail the two sub-routines, *computeMatrixPorilfe* and the *ComputeSubMP*.

4.3 Computing The Matrix Profile

The routine *ComputeMatrixProfile* (Algorithm 3) computes a matrix profile for a given subsequence length, ℓ. It essentially follows the STOMP algorithm [56], except that we also calculate the lower bound distance profiles in line 18. In line 5, the dot product between the sequence $T_{1,\ell}$ and the others in T is computed in *frequency domain* in $O(nlogn)$ time, where $n = |T|$. The dot product is computed in constant time in line 11 by using the result of the previous overlapping subsequences.

In line 7 we measure each z-normalized Euclidean distance, between $T_{i,\ell}$ and the other subsequence of length ℓ in T, avoiding trivial matches. The distance measure formula utilized is the following [30, 53, 56]:

$$dist(T_{i,\ell}, T_{j,\ell}) = \sqrt{2\ell(1 - \frac{QT_{i,j} - \ell\mu_i\mu_j}{\ell\sigma_i\sigma_j})} \quad (3)$$

In Eq. 3 $QT_{i,j}$ represents the dot product of the two sub-series with offset i and j respectively. It is important to note that, we may compute μ and σ in constant time by using the *running* plain and squared sum, namely s and ss (initialized in line 6). It follows that $\mu = s/\ell$ and $\sigma = \sqrt{(ss/\ell) - \mu^2}$.

In lines 8 and 16, we update both the matrix profile and the matrix profile index, which holds the offset of the closest match for each $T_{i,1}$.

Algorithm 3 ends with the loop in line 18, which evaluates the lower bound distance profile and stores the p smallest lower bound distance values in *listDP*. In line 19, the procedure *compLB* evaluates the lower bound distance profile introduced in Section 4.1 using Eq. 2. The structure *listDP* is a Max Heap with a maximum capacity of p. Each entry e of the distance profile in line 18 is a tuple containing the Euclidean distance between a subsequence $T_{j,\ell}$ and its nearest neighbor, the location of that nearest neighbor, the lower bound Euclidean distance of the pair, the dot product of them, and the plain and squared sum of $T_{j,\ell}$. In Figure 6 (b) we show an example of the distance profile in line 18. The distance profile is sorted according to the lower bound Eulidean distance values (shown as LB in the figure). The entries corresponding to the p smallest LB values are stored in memory to be reused for longer motif lengths.

The overall time complexity of the *ComputeMatrixProfile* routine is $O(n^2 logp)$, where n is the number of the computed distance profiles.

4.4 Matrix Profile for Subsequent Lengths

We are now ready to describe our *ComputeSubMP* algorithm, which allows us to find the motifs for subsequence lengths greater than ℓ in linear time.

The input of *ComputeSubMP*, whose pseudo-code is shown in Algorithm 4, is the vector *listDp* that we built in the previous step. In line 5, we start to iterate over the $p \times n$ elements of *listDp* in order to find the motif pair of length *newL*, using a procedure that is faster than Algorithm 1, leading to a complexity that is now linear in the best case (as the experiments show, this is often the case). Since *listDP* potentially contains enough elements to compute the whole matrix profile, it can provide more information than just the motif pair.

In the loop of line 9, we update all the entries of *listDP*[i] by computing the Euclidean and lower bound distance for the length *newL*. This operation is valid, since the ranking of each *listDP*[i] is maintained as the lower bound gets updated. Moreover, this latter computation is done in constant time (line 10), since the entries contain the statistics (i.e. sum, squared sum, dot product) for the length *newL*−1. Also note that the routine *updateDistAndLB* avoids the trivial matches, which may result from the length increment.

Subsequently, the algorithm checks in line 16 if *minDist* is smaller than or equal to *maxLB*, the largest lower bound distance value in *listDP*[i]. If this is true, *minDist* is the smallest value in the whole distance profile. In lines 17 and 18, we update the best-so-far distance value and the matrix profile. On the other hand, we update the smallest max lower bounding distance in line 21, recording also that we do not have the true min for the distance profile with offset i (line 23). Here, we may also note that even though the local true min is larger than the max lower bound (i.e., the condition of line 16 is not true), *minDist* may still represent an approximation of the true matrix profile point.

Algorithm 4: *ComputeSubMP*

```
   Input: DataSeries T, int nDp, MaxHeap[] listDP, int newL, int p
   Output: bBestM, SubMP, IP
1  double[] SubMP ← double[nDp];
2  int[] IP ← int[nDp];
3  double minDistAbs ← inf, double minLbAbs ← inf;
4  List ⟨ int,double ⟩ nonValidDP;
   // iterate over the partial distance profiles in listDP
5  for i ← 1 to nDp do
6      double minDist ← inf;
7      int ind ← 0;
8      double maxLB ← popMax(listDP[i]);
       // update the partial distance profile for the length newL (true
          Euclidean and lower bounding distance )
9      for each entry e in listDP[i] do
10         e.dist, e.LB ← updateDistAndLB(e, newL);
11         minDist ← min(minDist,e.dist));
12         if minDist == e.dist then
13             ind = e.offset;
14         end
15     end
       // check if the min (minDist) of this partial distance profile is the
          min of the complete distance profile
16     if minDist < maxLB then
           // minDist is the real min; valid distance profile
17         minDistABS ← min(minDistAbs,minDist);
18         SubMP[i] = minDist;
19         IP[i] =ind;
20     else
           // minDist is not the real min; non-valid distance profile
21         minLbAbs ← min(minLbAbs, maxLB));
22         SubMP[i] = ⊥;
23         nonValidDP.add((i, maxLB))
24     end
25 end
26 bool bBestM ← (minDistABS < minLbAbs) ;
   // if SubMP does not contain the motif distance (bBestM = false), compute
      the whole non valid distance profiles, if it is faster then
      computeMatrixProfile (nDp / p = true)
27 if !bBestM and nonValidDP.size() < (nDp/p) then
28     for each pair < ind, lbMax > in nonValidDP do
29         if lbMax < minDistABS then
30             QT ← SlidingDotProduct(T_{ind,ℓ}, T);
31             double s ← sum(T_{ind,ℓ}); double ss ← squaredSum(T_{ind,ℓ});
32             D(T_{ind,ℓ}) ← CalcDistProf(QT,T_{ind,ℓ}, T, s, ss);
33             SubMP[ind], IP[ind] = min(D(T_{ind,ℓ}));
34             insert(listDP[ind], D(T_{ind,ℓ}));
35         end
36     end
37     bBestM ← 1;
38 end
```

When the iteration of the partial distance profiles ends (end of for loop in line 5), the algorithm has enough elements to know if the matrix profile computed contains the real motif pair. In line 26, we verify if the smallest Euclidean distance we computed (*minDistABS*) is less than *minLbAbs*, which is the minimum lower bound of the *non-valid* distance profiles. We call non-valid, all the partial distance profiles, for which the maximum lower bound distance (i.e., the p-th largest lower bound of the distance profile) is smaller than the minimum true distance (line 20); otherwise, we call them valid (line 16).

As a result of the ranking preservation of the lower bounding function, if the above criterion holds, we know that each true Euclidean distance in the non-valid distance profiles must be greater than *minDistABS*. In line 27, the algorithm has its last opportunity to exploit the lower bound in the distance profiles, in order to avoid computing the whole matrix profile. If *bBestM* is false (the motif has not been found), we start to iterate through the non-valid

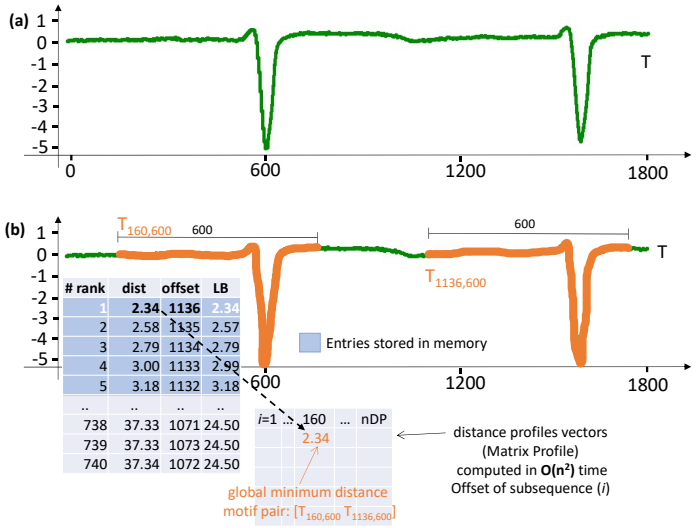

Figure 6: (*a*) Input time series, (*b*) Compute matrix profile snapshot: (on the left) distance profile of the subsequence $T_{160,600}$ which is part of the motif.

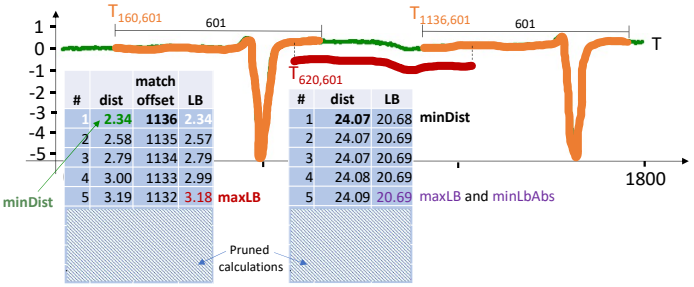

Figure 7: Compute Sub Matrix profile: the partial distance profile of $T_{160,601}$ contains the motif's subsequences distance.

distances profiles. Note that we perform this iteration when their number is less than half of the total distance profiles.

We present here two examples that explain the main procedures of *VALMOD*.

Example 4.1. In Figure 6, we show a snapshot of a VALMOD run. In Figure 6(a), VALMOD receives as input a data series of length 1800. In Figure 6(b), the matrix profile for subsequence length $\ell = 600$ is computed (Algorithm 3). On the left, we depict the distance profile regarding $T_{160,600}$, and rank it according to the lower bound (LB) distance values. Although we are computing the entire distance profile, we store only the first $p = 5$ entries in memory.

Example 4.2. Figure 7 shows the execution of *ComputeSubMP* (Algorithm 4), taking place after the step illustrated in Figure 6(b). In this picture, we show the distance profile of a subsequence belonging to the motif pair, for subsequence length $\ell = 601$. This time it is built by computing $p = 5$ distances (left side of the picture). We can now make the following observations.

(a) In the distance profile of the subsequence $T_{160,601}$ (left array): $minDist = 2.34 < maxLB = 3.18 \iff$ the value 2.34 is both a local and a global minimum (among all the distance profiles).
(b) Considering the partial distance profile of subsequence $T_{620,601}$ (right array), we do not know if its *minDist* is its real global minimum, since 20.69 (*maxLB*) < 24.07 (*minDist*).
(c) We know, that 20.69 (*maxLB* of the distance profile of subsequence $T_{620,601}$) is the *minLbAbs*, or in other words, the smallest *maxLB* distance among all the partial distance profiles in which $maxLB < minDist$ holds.
(d) We know that there are no true Euclidean distances (among those computed) smaller than 2.34. Since $minDist = 2.34 < minLbAbs = 20.69 \iff$ 2.34 is the distance of the motif $\{T_{160,601}; T_{1136,601}\}$.

In the best case, *ComputeSubMP* takes $O(np)$ time, whereas the worst case time complexity is $O(nClogn)$, where n is the number of the computed distance profiles, and $C = n/p$.

5 FINDING MOTIF SETS

We finally extend our technique in order to find the variable-length motif sets. In that regard, we start to consider the top-K motif pairs, namely the pairs having the K smallest length-normalized distances. The idea is to extend each motif pair to a motif set considering the subsequences proximity as a quality measure, favoring thus the motif sets, which contain the closest subsequence pairs. Moreover, for each top-K motif pair ($T_{a,\ell}$,$T_{b,\ell}$), we use a radius $r = D * dist(T_{a,\ell}, T_{b,\ell})$, when we extend it to a motif set. We call the real variable D *radius factor*. This choice, permits to tune the radius r by the user defined radius factor, considering also the characteristics of the data. Setting a unique and non data dependent radius for all motif sets, would penalize the results of exploratory analysis.

First, we introduce Algorithm 5, a slightly modified version of the *updateValmp* routine (Algorithm 2). The new algorithm is called *updateVALMPForMotifSets*, and its main goal is to keep track of the best K subsequence pairs (motif pairs) according to the *VALMP* ranking, and the corresponding partial distance profiles. The idea is to later exploit the lower bounding distances for pruning computations, while computing the motif sets.

In lines 4 to 7, we build a structure named *pair*, which carries the information of the subsequences pairs that appear in the *VALMAP* structure. During this iteration, we leave the fields *partDP*1 and *partDP*2 empty, since they will be later initialized with the partial distance profiles, if their *pair* is in the top K of *VALMAP*. In order to enumerate the best K pairs, we use the global maximum heap *heapBestKPairs* in line 8. Then, we assign to each pair (or update) the corresponding partial distance profiles (line 15).

We are now ready to present the variable length motif sets discovery algorithm (refer to Algorithm 6). Starting at line 1, the algorithm iterates over the best pairs. For each one of those, we need to check if the search range is smaller than the maximum lower bound distances of both partial distance profiles. If this is true, we are guaranteed that we have already computed all the subsequences in the range. Therefore, in lines 7 and 14 we filter the subsequences in the range, sorting the partial distance profile according to the offsets. This operation will permit to find the trivial matches in linear time.

Algorithm 5: *updateVALMPForMotifSets*

```
   Input: VALMP, double [] MPnew, int [] IP, nDP, ℓ, MaxHeap[] listDP,
   Output: VALMP
1  for i ← 1 to nDP do
       // length normalize the Euclidean distance
2      double lNormDist ← MPnew[i] * sqrt(1/ℓ);
       // if the distance at offset i of VALMP, surely comptued with previous
          lengths, is larger than the actual, update it
3      if (VALMP.distances[i] > lNormDist or VALMP.MP[i] == ⊥) then
4          entry pair;
5          pair.off1 ← i, pair.off2 ← IP[i];
6          pair.distance ← MPnew[i], pair.ℓ ← ℓ;
7          pair.partDP1 ← ⊥, pair.partDP2 ← ⊥;
8          insert(heapBestKPairs, pair);
9          VALMP.distances[i] ← MPnew[i];
10         VALMP.normDistances[i] ← lNormDist;
11         VALMP.lengths[i] ← ℓ;
12         VALMP.indices[i] ← IP[i];
13     end
14 end
15 for each pair in heapBestKPairs do
16     if (pair.partDP1== ⊥) then
17         pair.partDP1 ← listDP[pair.off1];
18         pair.partDP2 ← listDP[pair.off2];
19     end
20 end
```

Algorithm 6: *computeVarLengthMotifSets*

```
   Input: DataSeries T, MaxHeap heapBestKPairs, int D
   Output: Set S*
1  for each pair in heapBestKPairs do
2      double r ← pair.distance * D;
3      double maxLB1 ← popMax(pair.partDP1);
4      double maxLB2 ← popMax(pair.partDP2);
5      D(T_{pair.off1,pair.ℓ}) ← ∅, D(T_{pair.off2,pair.ℓ}) ← ∅;
6      if maxLB1 > r then
           // sort according the offset, the partial distance profile
              contains all the elements in the range
7          D(T_{pair.off1,pair.ℓ}) ←
             sortAndFilterRange(r,pair.partDP1.toVector());
8      else
           // re-compute the mat
9          double s ← sum(T_{ind,ℓ});
10         double ss ← squaredSum(T_{ind,ℓ});
11         D(T_{pair.off1,pair.ℓ}) ←
             CalcDistProfInRange(r,QT,T_{pair.off1,pair.ℓ}, T, s, ss);
12     end
13     if maxLB2 > r then
14         D(T_{pair.off2,ℓ}) ← sortAndFilterRange(r,pair.partDP2.toVector());
15     else
16         double s ← sum(T_{ind,ℓ});
17         double ss ← squaredSum(T_{ind,ℓ});
18         D(T_{pair.off2,pair.ℓ}) ←
             CalcDistProfInRange(r,QT,T_{pair.off1,pair.ℓ}, T, s, ss);
19     end
20     Set S_r^{pair.ℓ} ← mergeRemoveTM(D(T_{pair.off1,ℓ}), D(T_{pair.off2,ℓ}));
21     S*.add(S_r^{pair.ℓ});
22 end
```

On the other hand, if the search range is larger than the maximum lower bound distances of both partial distance profiles, we have to re-compute the entire distance profile (lines 11 and 18), to find all the subsequences in the range. Once we have the distance profile pairs we need to merge them and remove the trivial matches (line 20). Each time we add a subsequence in a motif set we remove it from the search space: this guarantees the empty intersection among the sets in S^*.

The complexity of the *updateVALMPForMotifSets* algorithm is $O(nlogK)$, while *computeVarLengthMotifSets* (i.e., the final algorithm) runs in $O(Kplogp)$ time.

6 EXPERIMENTAL EVALUATION

6.1 Setup

Configuration. We implemented our algorithms in C (compiled with gcc 4.8.4), and we ran them in a machine with the following hardware: Intel Xeon E3-1241v3 (4 cores - 8MB cache - 3.50GHz - 32GB of memory). All the experiments in this paper are completely reproducible. In that regard, the interested reader can find the analyzed datasets and source code in the paper web page [21].

Datasets And Benchmarking Details. To benchmark our algorithm, we use five different datasets:

- (GAP), which contains the recording of the global active electric power in France for the period 2006-2008. This dataset is provided by EDF (main electricity supplier in France) [20];
- (CAP), the Cyclic Alternating Pattern dataset, which contains the EEG activity occurring during NREM sleep phase [7];
- (ECG) and (EMG) signals from Stress Recognition in Automobile Drivers [12];
- (ASTRO), which contains a data series representing celestial objects [41].

Table 1 summarizes the characteristics of the real datasets we used in our experimental evaluation. For each dataset, we report the minimum and maximum values, the overall mean and standard deviation, and the total number of points.

	MIN	MAX	MEAN	STD-DEV	number of points
ECG	-2.182	1.543	0.006	0.24	1M
GAP	0.08	10.67	1.10	1.15	2M
ASTRO	-0.00867	0.00447	0.00003	0.00031	2M
EMG	-0.694	0.773	-0.005	0.041	1M
EEG	-966	920	3.34	41.36	0.5M

Table 1: Characteristics of the datasets used in the experimental evaluation.

The (CAP),(ECG) and (EMG) datasets are available in [10]. We use several prefix snippets of these datasets, ranging from 0.1M to 1M of points.

In order to measure the scalability of our approach, we test its performance along four dimensions, which are depicted in Table 2. Each experiment is conducted by varying the parameter of a single column, while for the others, the default value (in bold) is selected. In our benchmark, we have two types of algorithms to compare to VALMOD. The first are two state-of-the-art motif discovery algorithms, which receive a single subsequence length as input: QUICKMOTIF [19] and STOMP [53]. In our experiments, they have been adapted to find all the motifs for a given subsequence length range. The other approach in the comparative analysis is

Motif length (ℓ_{min})	Motif range ($\ell_{max}-\ell_{min}$)	Data series size (points)	p (elements of distance profiles stored)
256	**100**	0.1 M	5
512	150	0.2 M	10
1024	200	**0.5 M**	15
2048	400	0.8 M	20
4096	600	1 M	**50** , 100 , 150

Table 2: Parameters of VALMOD benchmarking (default values shown in bold).

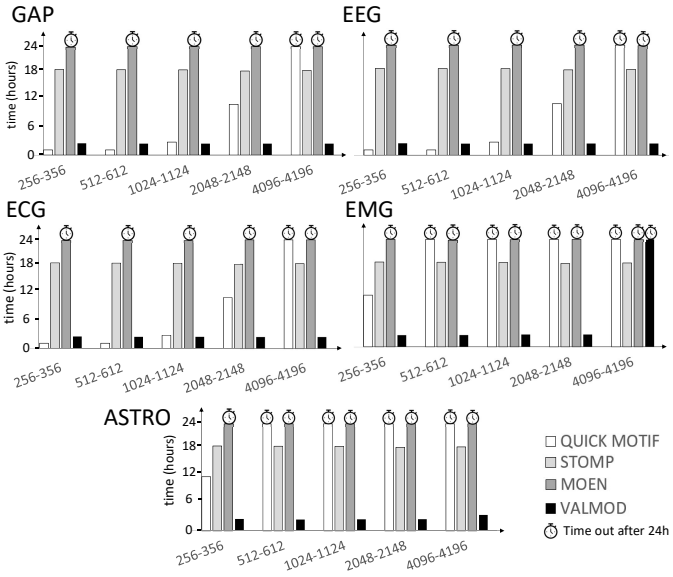

Figure 8: Scalability for various motif length ranges.

MOEN [29], which accepts a range of lengths as input, producing the best motif pair for each length.

6.2 Results

For VALMOD, we report the total time, including the time to build the matrix profile (Algorithm 3).

Scalability Over Motif Length. In Figure 8, we depict the performance results of the *four* motif discovery approaches, when varying the motif length. We note that the performance of VALMOD remains stable over the five datasets. On the other hand, we observe that a pruning strategy based on a summarized version of the data is sensitive to subsequence length variation. This is the case for QUICK MOTIF, which operates on PAA (Piecewise Aggregate Approximation) discretized data. Figure 8 shows that the performance of QUICK MOTIF varies significantly as a function of the motif length range, growing rapidly as the range increases, and failing to finish within a reasonable amount of time in several cases.

Moreover, we argue that our proposed lower bounding measure enables our method to improve upon MOEN, which clearly does not scale well in this experiment (see Figure 8). The main reason of

this behavior is that the effectiveness of the lower bound of MOEN decreases very quickly as we increase the subsequence length ℓ. When we increase the subsequence length by 1, MOEN multiplies the lower bound by a value smaller than 1 ([29], Section IV.B), thus making it less tight. In contrast, the lower bound of VALMOD does not always decrease (refer to Eq. 2): $\frac{\sigma_{j,l}}{\sigma_{j,l+k}}$ may be larger than 1. Consequently, the lower bound of VALMOD can remain effective (i.e., tight) even after several steps of increasing the subsequence length.

Concerning the VALMOD performance, we note a sole exception that appears in the noisy EMG data (bottom of Figure 8), for a relatively high motif length range (4096-4196). A possible explanation for this behavior is that the lower bounding distance used by VALMOD is coarse, or in other words, it is not a good approximation of the true distance. Figure 9 shows the difference between the greater lower bounding distance (*maxLB*) and the smaller true Euclidean distance for each distance profile. We use the subsequence lengths *356* and *4196*, which are the extremes for this experiment. In this last plot, each value greater than 0 corresponds to a valid condition in line 16 of the *ComputeSubMP* algorithm. This indicates that we found the smallest value of a distance profile, while pruning computations. As the subsequence length increases, VALMOD's pruning becomes less effective for the EMG dataset. On the other hand, it shows no variability of its pruning effectiveness over the ECG dataset. The Tightness of the Lower Bound (TLB) [43, 58] is a measure of the Lower bounding quality, which given two data series t_1 and t_2, is computed as follows: $LB_{dist}(t_1, t_2)/EuclideanDistance(t_1, t_2)$ (note that TLB ranges from 0 to 1).

In Figure 10, we show the average TLB for each (partial) distance profile. In the EMG dataset, when using the larger subsequence length, we observe a sharp decrease of the lower bounding quality (small TLB), explaining the behavior observed in Figure 9(left). On the other hand, for all the subsequence lengths in the ECG dataset, we have almost the same TLB (Figure 10(right)), explaining the stable performance of VALMOD on this dataset.

In Figure 11, we also show the distribution of the pairwise subsequences distance, using the same datasets and lengths. Note that here we plot the Euclidean distance without length normalization, since the algorithm uses it to rank the motifs in the trailing part. In the EMG dataset, as the length increases, the distance distribution includes many high values, which affect VALMOD negatively. Observe that in the ECG dataset, the values are more uniformly distributed over all the subsequence lengths.

Note that in the discussion above, we have used the results of the ECG and EMG datasets only, which represent the extreme cases for VALMOD, exhibiting the overall best and worst performance, respectively. The rest of the datasets produce results that lie between these two extremes, and are omitted for brevity.

Scalability Over Motif Range. In Figure 12, we depict the performance results as the motif range increases. VALMOD gracefully scales on this dimension, whereas the other approaches can seldom complete the task. Not only does our technique address the intrinsic problem of STOMP and QUICK MOTIF, which independently process each subsequence length, but it also exhibits a substantial

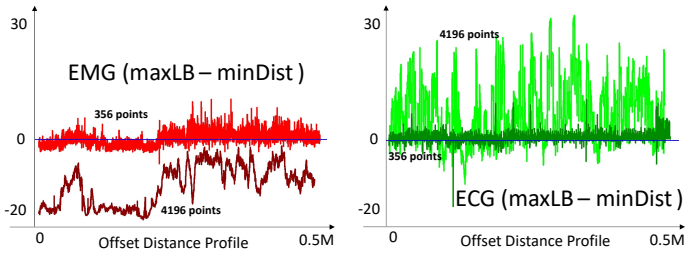

Figure 9: The difference between the max lower bounding distance (maxLB) and the min Euclidean distance of partial distance profiles (ECG and EMG). Subsequence lengths: 356/4196.

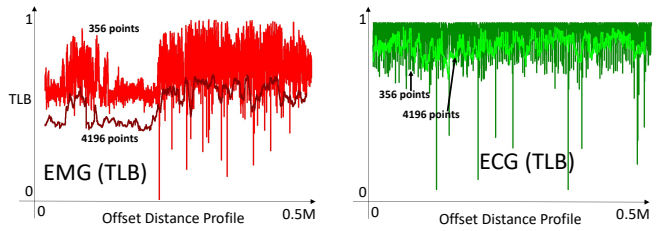

Figure 10: Average of the tightness of the lower bounding (TLB) for every Distance profile of dataset EMG and ECG for the subsequence lengths: 356/4196.

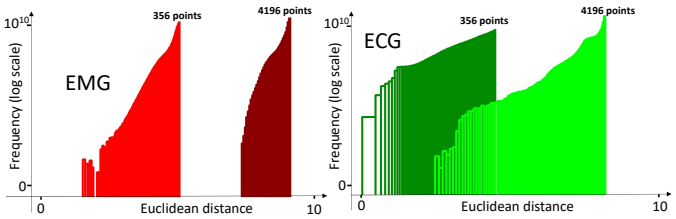

Figure 11: Distribution of Euclidean distance of pairwise subsequences of ECG and EMG datasets.

improvement over MOEN, the existing state-of-the-art approach for discovery of variable length motifs.

Scalability Over Data Series Length. In Figure 13, we experiment with different data series sizes. For the EEG dataset we only report three measurements, since this collection contains no more than 0.5M points. We observe that QUICK MOTIF exhibits high sensitivity, not only to the various data sizes, but also to the different datasets (as in the previous case, where we varied the subsequence length). It is also interesting to note that QUICK MOTIF is sligthly faster than VALMOD on the ECG dataset, which contains regular and similar heartbeat patterns, and is a relatively easy dataset for motif discovery. Nevertheless, QUICK MOTIF, as well STOMP and MOEN, fail to terminate within a reasonable amount of time for the majority of our experiments. On the other hand, VALMOD does not exhibit any abrupt changes in its performance, scaling gracefully with the size of the dataset, across all datasets and sizes.

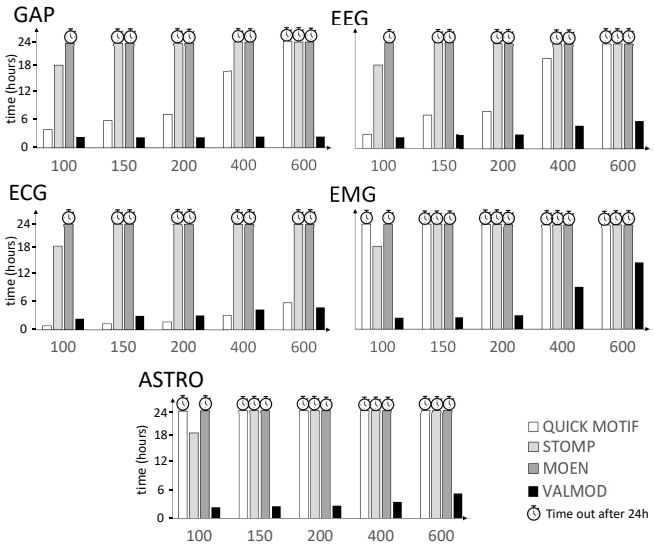

Figure 12: Scalability with increasing motif range.

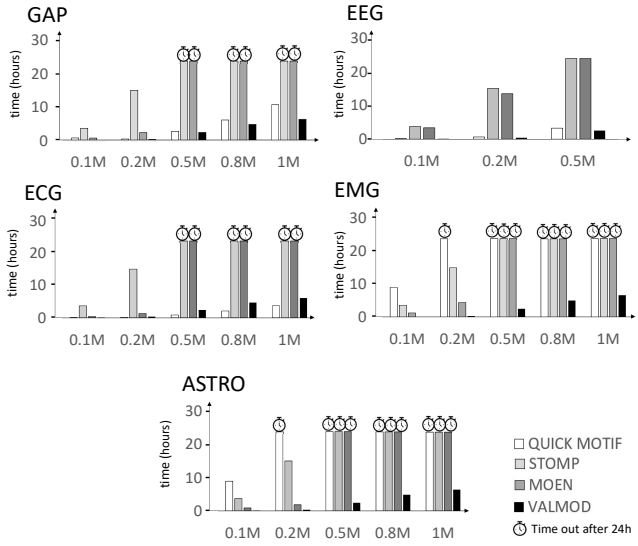

Figure 13: Scalability with increasing data series size.

Effect of Changing Parameter p. In Figure 14, we study the effect of parameter p on VALMOD's performance. The p value determines how many distance profile entries we compute and keep in the memory. Increasing p leads to increased memory consumption, but could also translate to an overall speed-up, since having more distances may guarantee a larger margin between the greater lower bounding distance and the minimum true Euclidean distance in a distance profile. As we can see in the left side of the plot, increasing p does not provide any significant advantage in terms of time complexity. Moreover, the plots on the right-hand side of the figure demonstrate that the size of the Matrix profile subset (*subMP*), computed by the *computeSubMP* procedure, decreases in the same manner at each iteration (i.e., as we increase the length of

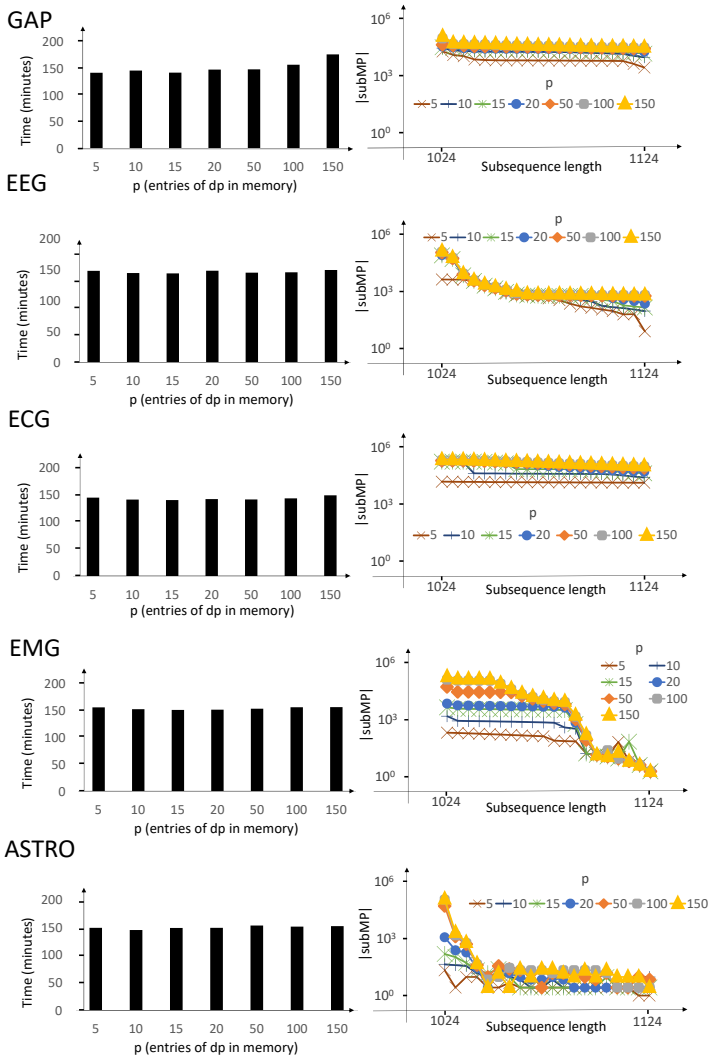

Figure 14: Scalability with increasing parameter p.

the subsequences that the algorithm considers), regardless of the value of p.

It is important to note that irrespective of its size, *subMP always* contains the smallest distances of the matrix profile, namely the distances of the motif pair. Having a larger *subMP* does not represent an advantage w.r.t motif discovery, but rather an opportunity to view and analyze the subsequence pairs, whose distances are close to the motif.

6.3 Motif Sets

We now conduct an experiment to show the time performance of identifying the variable length motif sets. We use the default values of Table 2, varying K and the radius factor D for each dataset. In Figure 15 we report the results; we also show the time to compute *VALMP* (the output of VALMOD). We note that once we build the pairs ranking of *VALMP* (*heapBestKPairs* in Algorithm 5), we can run the procedure that computes the motif sets (Algorithm 6). The results show that this operation is 3-6 orders of magnitude

	GAP		EEG		ECG		EMG		ASTRO	
	VALMP time	9601 seconds	VALMP time	9608 seconds	VALMP time	9653 seconds	VALMP time	10294 seconds	VALMP time	9100 seconds
a)	K	Top K sets (seconds)	K	Top K sets (seconds)	K	Top K sets (seconds)	K	Top K sets (seconds)	K	Top K sets (seconds)
	10	1.74	**10**	0.09	**10**	0.001	**10**	1.64	**10**	1.60
	20	3.37	**20**	0.09	**20**	0.001	**20**	3.27	**20**	3.23
	40	6.66	**40**	0.09	**40**	0.001	**40**	6.53	**40**	6.37
	60	10	**60**	0.09	**60**	0.001	**60**	9.81	**60**	9.44
	80	13.33	**80**	0.09	**80**	0.001	**80**	13.08	**80**	12.59
b)	D	Top K sets (seconds)	D	Top K sets (seconds)	D	Top K sets (seconds)	D	Top K sets (seconds)	D	Top K sets (seconds)
	2	0.0015	**2**	0.001	**2**	0.001	**2**	6.52	**2**	4.45
	3	0.0016	**3**	0.001	**3**	0.001	**3**	6.50	**3**	5.79
	4	6.67	**4**	0.22	**4**	0.001	**4**	6.88	**4**	6.12
	5	6.67	**5**	0.35	**5**	0.001	**5**	7.47	**5**	6.45
	6	6.67	**6**	0.88	**6**	0.001	**6**	6.86	**6**	6.56

Figure 15: Time performance of variable length motif sets discovery. (a) Varying K (default D=4). (b) Varying radius factor D (default K=40).

faster than the computation of *VALMP*. The advantage in time performance is pronounced for the *ECG* and *EEG* datasets, thanks to the pruning we perform with the partial distance profiles.

The fast performance of the proposed approach also allows for a fast exploratory analysis over the radius factor, which would otherwise (i.e., with previous approaches) be extremely time-consuming to set for each dataset.

7 RELATED WORK

While research on data series similarity measures and data series query-by-content [34, 36] has a very long history, *data series motifs* were introduced just fifteen years ago [5]. Following their introduction, there was an explosion of interest in their use for diverse applications. There exist analogies between *data series motifs* and sequence *motifs* (in DNA), which have been exploited. For example, discriminative motifs in bioinformatics [44] inspired discriminative data series motifs (i.e., data series shapelets) [32]. Likewise, the work of Grabocka et al. [11] on generating idealized motifs, is similar to the idea of consensus sequence (or canonical sequence) in molecular biology. The literature on the general data series motif search is vast; we refer the reader to recent studies [53, 56] and their references.

The QUICK MOTIF [19] and STOMP [56] algorithms represent the state of the art for fixed-length motif pair discovery. QUICK MOTIF first builds a summarized representation of the data using Piecewise Aggregate Approximation (PAA), and arranges these summaries in Minimum Bounding Rectangles (MBRs) in a Hilbert R-Tree index. The algorithm then prunes the search space based on the MBRs. On the other hand, STOMP is based on the computation of the matrix profile, in order to discover the best matches for each subsequence. The smallest of these matches is the motif pair. We observe that both the above approaches solve a restricted version of our problem: they discover motif sets of cardinality two (i.e., motif pairs) of a fixed, predefined length. On the contrary, VALMOD removes these limitations and proposes a general and efficient solution. Its main contributions are the novel algorithm for examining candidates of various lengths and corresponding lower bounding distance: these techniques help to reuse the computations performed so far, and lead to effective pruning of the vast search space.

We note that there are only three studies that deal with issues of variable length motifs, and attempt to address them [8, 25, 54]. While these studies are pioneers in demonstrating the *utility* of variable length motifs, they cannot serve as practical solutions to the task at hand for two reasons: (i) they are all approximate, while we need to produce exact results; and (ii) they require setting many parameters (most of which are unintuitive). Approximate algorithms can be very useful in many contexts, if the amount of error can be bounded, or at least known. However, this is not the case for the algorithms in question. In certain cases, such as when analyzing seismological data, the threat of litigation, or even criminal proceedings [4], would make any analyst reluctant to use an approximate algorithm.

The other work to explicitly consider variable length motifs is MOEN [29]. Its operation is based on the distance computation of subsequences of increasing length, and a corresponding pruning strategy based on upper and lower bounds of the distance computed for the smaller length subsequences. Unlike the algorithms discussed above, MOEN is exact and requires few parameters. However, it has been tuned for producing only a single motif pair for each length in the range, and as our evaluation showed, it is not competitive in terms of time-performance to our approach. This is due to its relatively loose lower bound and sub-optimal search space pruning strategy, which force the algorithm to perform more work than necessary.

In summary, while there is a large and growing body of work that both uses motif discovery in diverse domains, or offers enhancements/generalizations of basic motif search, this work offers the first scalable, parameter-light, *exact* variable-length motif set discovery algorithm in the literature.

8 CONCLUSIONS

Motif discovery is an important subject of inquiry in data series mining across several domains, and a key operation necessary for several analysis tasks. Even though much effort has been dedicated to this problem, no solution had been proposed for discovering motifs of different lengths.

In this work, we propose the first algorithm for variable-length motif discovery. We describe a new distance normalization method, as well as a novel distance lower bounding technique; both of which are necessary for the solution to our problem. We experimentally evaluated our algorithm by using five real datasets from diverse domains. The results demonstrate the efficiency and scalability of our approach (up to 20x faster than the state of the art), also its usefulness.

In terms of future work, we would like to further improve the pruning effectiveness of VALMOD, which would lead to faster performance even for larger motif length ranges. We also plan to extend VALMOD in order to efficiently compute a complete matrix profile for each length in the input range. This would enable us to support more diverse applications, such as discovery of *shapelets* [52] and *discords* [23, 53].

REFERENCES

[1] Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. 1993. Efficient Similarity Search In Sequence Databases. In *FODO '93*. 69–84.

[2] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. 2010. iSAX 2.0: Indexing and mining one billion time series. In *ICDM*.

[3] Alessandro Camerra, Jin Shieh, Themis Palpanas, Thanawin Rakthanmanon, and Eamonn J. Keogh. 2014. Beyond one billion time series: indexing and mining very large time series collections with iSAX2+. *KAIS* 39, 1 (2014), 123–151.

[4] Edwin Cartlidge. Oct. 3, 2016. Seven-year legal saga ends as Italian official is cleared of manslaughter in earthquake trial. *Science* (Oct. 3, 2016).

[5] Bill Yuan Chiu, Eamonn J. Keogh, and Stefano Lonardi. 2003. Probabilistic discovery of time series motifs. In *SIGKDD 2003*. 493–498.

[6] Michele Dallachiesa, Themis Palpanas, and Ihab F. Ilyas. 2014. Top-k Nearest Neighbor Search In Uncertain Data Series. *PVLDB* 8, 1 (2014), 13–24.

[7] MG Terzano et al. Sleep Med 2001 Nov, 2(6):537-553. Atlas, rules, and recording techniques for the scoring of cyclic alternating pattern (CAP) in human sleep. (Sleep Med 2001 Nov, 2(6):537-553).

[8] Yifeng Gao, Jessica Lin, and Huzefa Rangwala. 2016. Iterative Grammar-Based Framework for Discovering Variable-Length Time Series Motifs. In *ICMLA 2016*. 7–12.

[9] C. Gisler, A. Ridi, D. Zufferey, O. A. Khaled, and J. Hennebert. 2013. Appliance consumption signature database and recognition test protocols. In *2013 WoSSPA)*. 336–341.

[10] Glass L Hausdorff JM Ivanov PCh Mark RG Mietus JE Moody GB Peng C-K Stanley HE. Goldberger AL, Amaral LAN. 2000 June 13. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. (2000 June 13). http://circ.ahajournals.org/cgi/content/full/101/23/e215

[11] Josif Grabocka, Nicolas Schilling, and Lars Schmidt-Thieme. 2016. Latent Time-Series Motifs. *TKDD* 11, 1 (2016), 6:1–6:20.

[12] Picard RW. Healey JA. June 2016. Detecting stress during real-world driving tasks using physiological sensors. *IEEE Transactions in Intelligent Transportation Systems 6(2):156-166* (June 2016).

[13] H. V. Jagadish, Alberto O. Mendelzon, and Tova Milo. 1995. Similarity-Based Queries. In *ACM SIGACT-SIGMOD-SIGART Symposium*.

[14] Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. 2017. Time Series Management Systems: A Survey. *IEEE Trans. Knowl. Data Eng.* 29, 11 (2017), 2581–2600.

[15] Shrikant Kashyap and Panagiotis Karras. 2011. Scalable kNN search on vertically stored time series. In *KDD*.

[16] Eamonn J. Keogh. 2011. Machine Learning in Time Series Databases (tutorial).

[17] Eamonn J. Keogh, Kaushik Chakrabarti, Sharad Mehrotra, and Michael J. Pazzani. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. In *ACM SIGMOD 2001*.

[18] Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. 2018. Coconut: A Scalable Bottom-Up Approach for Building Data Series Indexes. In *PVLDB*.

[19] Yuhong Li, Leong Hou U, Man Lung Yiu, and Zhiguo Gong. 2015. Quick-motif: An efficient and scalable framework for exact motif discovery. (2015), 579–590 pages.

[20] M. Lichman. 2013. UCI Machine Learning Repository. (2013). http://archive.ics.uci.edu/ml

[21] Michele Linardi. 2017. VALMOD support web page. (2017). http://www.mi.parisdescartes.fr/~mlinardi/VALMOD.html

[22] Michele Linardi and Themis Palpanas. 2018. ULISSE: ULtra compact Index for Variable-Length Similarity SEarch in Data Series. In *ICDE*.

[23] Wei Luo, Marcus Gallagher, and Janet Wiles. 2013. Parameter-Free Search of Time-Series Discord. *Journal of Computer Science and Technology* (2013).

[24] A. Marzal and E. Vidal. 1993. Computation of Normalized Edit Distance and Applications. *IEEE Trans. Pattern Anal. Mach. Intell.* 15, 9 (Sept. 1993).

[25] David Minnen, Charles Lee Isbell Jr., Irfan A. Essa, and Thad Starner. Discovering Multivariate Motifs using Subsequence Density Estimation and Greedy Mixture Learning. In *AAAI Conference on Artificial Intelligence, 2007*.

[26] Katsiaryna Mirylenka, Vassilis Christophides, Themis Palpanas, Ioannis Pefkianakis, and Martin May. Characterizing Home Device Usage From Wireless Traffic Time Series. In *EDBT, 2016*.

[27] Y. Mohammad and T. Nishida. 2012. Unsupervised discovery of basic human actions from activity recording datasets. In *2012 IEEE/SICE International Symposium on System Integration (SII)*.

[28] Yasser F. O. Mohammad and Toyoaki Nishida. Exact Discovery of Length-Range Motifs. In *Intelligent Information and Database Systems - 6th Asian Conference, ACIIDS 2014*.

[29] Abdullah Mueen. Enumeration of Time Series Motifs of All Lengths. In *ICDM, 2013*.

[30] Abdullah Mueen, Hossein Hamooni, and Trilce Estrada. 2014. Time Series Join on Subsequence Correlation. In *2014 IEEE International Conference on Data Mining, ICDM 2014, Shenzhen, China, December 14-17, 2014*. 450–459.

[31] Abdullah Mueen, Eamonn J. Keogh, Qiang Zhu, Sydney Cash, and M. Brandon Westover. Exact Discovery of Time Series Motifs. In *SDM 2009*.

[32] Moss C. B. Neupane, D. and A. H. 2016. van Bruggen. 2016. Estimating citrus production loss due to citrus huanglongbing in Florida. *Annual Meeting, Southern Agricultural Economics Association, San Antonio, TX*. (2016).

[33] Michael. Noskov. 2015. Director, Data Science at Aspen Technology. Personal communication. (2015).

[34] Themis Palpanas. Big Sequence Management: A glimpse of the Past, the Present, and the Future. In *SOFSEM 2016*.

[35] Themis Palpanas. 2015. Data Series Management: The Road to Big Sequence Analytics. *SIGMOD Record* 44, 2 (2015), 47–52.

[36] Themis Palpanas. 2017. The Parallel and Distributed Future of Data Series Mining. In *High Performance Computing & Simulation (HPCS)*.

[37] Spiros Papadimitriou and Philip S. Yu. Optimal multi-scale patterns in time series streams. In *ACM SIGMOD 2006*.

[38] Davood Rafiei and Alberto Mendelzon. 1998. Efficient Retrieval of Similar Time Sequences Using DFT. In *ICDE*.

[39] Usman Raza, Alessandro Camerra, Amy L. Murphy, Themis Palpanas, and Gian Pietro Picco. 2015. Practical Data Prediction for Real-World Wireless Sensor Networks. *IEEE Trans. Knowl. Data Eng.* (2015).

[40] D. Roverso. 2000. Multivariate Temporal Classification by Windowed Wavelet Decomposition and Recurrent Networks. In *ANS International Topical Meeting on Nuclear Plant Instrumentation, Control and Human-Machine Interface*.

[41] W.H.Baumgartner G.Ponti C.R.Shrader P. Lubinski H.A.Krimm F. Mattana J. Tueller S. Soldi, V. Beckmann. 2014. Long-term variability of AGN at hard X-rays. *Astronomy & Astrophysics* (2014).

[42] Suchi Saria, Andrew Duchi, and Daphne Koller. Discovering Deformable Motifs in Continuous Time Series Data. In *IJCAI 2011*.

[43] Jin Shieh and Eamonn Keogh. 2008. iSAX: Indexing and Mining Terabyte Sized Time Series. In *SIGKDD*. 623–631.

[44] Saurabh Sinha. 2002. Discriminative motifs. In *Proceedings of the Sixth Annual International Conference on Computational Biology, RECOMB 2002*. 291–298.

[45] Zeeshan Syed, Collin M. Stultz, Manolis Kellis, Piotr Indyk, and John V. Guttag. 2010. Motif discovery in physiological datasets: A methodology for inferring predictive elements. *TKDD* 4, 1 (2010), 2:1–2:23.

[46] J. Wang, A. Balasubramanian, L. Mojica de la Vega, J. Green, A. Samal, and B. Prabhakaran. Word recognition from continuous articulatory movement time-series data using symbolic representations. In *Workshop on Speech and Language Processing for Assistive Technologies. (SLPAT)* (2013).

[47] Yang Wang, Peng Wang, Jian Pei, Wei Wang, and Sheng Huang. 2013. A Data-adaptive and Dynamic Segmentation Index for Whole Matching on Time Series. *PVLDB* (2013).

[48] CW Whitney, DJ Gottlieb, S Redline, RG Norman, RR Dodge, E Shahar, S Surovec, and FJ Nieto. 1998. Reliability of scoring respiratory disturbance indices and sleep staging. *Sleep* (November 1998).

[49] Denis S. Willett, Justin George, Nora S. Willett, Lukasz L. Stelinski, and Stephen L. Lapointe. 2016. Machine Learning for Characterization of Insect Vector Feeding. *PLOS Computational Biology* (2016).

[50] Djamel-Edine Yagoubi, Reza Akbarinia, Florent Masseglia, and Themis Palpanas. 2017. DPiSAX: Massively Distributed Partitioned iSAX.

[51] Dragomir Yankov, Eamonn J. Keogh, Jose Medina, Bill Yuan-chi Chiu, and Victor B. Zordan. Detecting time series motifs under uniform scaling. In *ACM*.

[52] Lexiang Ye and Eamonn J. Keogh. 2009. Time series shapelets: a new primitive for data mining. In *KDD*.

[53] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn J. Keogh. Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets. In *IEEE , ICDM 2016*.

[54] Sorrachai Yingchareonthawornchai, Haemwaan Sivaraks, Thanawin Rakthanmanon, and Chotirat Ann Ratanamahatana. 2013. Efficient Proper Length Time Series Motif Discovery. In *2013 IEEE ICDM*. 1265–1270.

[55] Yan Zhu, Abdullah Mueen, and Eamonn Keogh. 2018. Admissible Time Series Motif Discovery with Missing Data. (2018). arXiv:1802.05472

[56] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Brisk, and Eamonn J. Keogh. Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins. In *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*.

[57] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. 2016. ADS: the adaptive data series index. *VLDB J.* 25, 6 (2016), 843–866.

[58] Kostas Zoumpatianos, Yin Lou, Themis Palpanas, and Johannes Gehrke. 2015. Query Workloads for Data Series Indexes. In *ACM SIGKDD, 2015*. 1603–1612.

[59] Kostas Zoumpatianos and Themis Palpanas. 2018. Data Series Management: Fulfilling the Need for Big Sequence Analytics.. In *ICDE*.

9 APPENDIX

9.1 Experimental Case Study in Entomology

Insects that feed by ingesting plant fluids cause devastating damage to agriculture worldwide, primarily by transmitting pathogens of plants [49]. In recent years, the citrus industry of Brazil and Florida has been devastated by the Huanglongbing disease, also known as citrus greening disease [32]. The authors of this work span France, Italy, and Southern California. These three regions are expected to be the next regions to suffer from the economic impacts of this disease; thus, we attempt to understand and eventually control the main vector, the Asian citrus psyllid (*Diaphorina citri*). These insect vectors can acquire and transmit pathogens, causing infectious diseases by probing on host tissues and ingesting host fluids. As shown in Figure 16, the feeding processes of these insects can be recorded by gluing a thin wire to the insect's back, completing the circuit through a host plant, and then measuring fluctuations in voltage level to create an Electrical Penetration Graph (EPG). Finding repeated patterns in these data is a vital step in building dictionaries of behavior, which is important to understanding and (perhaps) ultimately controlling these pests.

We conducted an experiment on a dataset with 205,000 datapoints (about 5.5 hours) of an Asian citrus psyllid feeding on a Valencia orange tree (*Citrus sinensis*), which was first hybridized in Southern California. We searched for motifs in the range of *10* to *12* seconds. As Figure 1 shows, over this range the top motif changes from the insect's highly technical probing skill as it searches for a rich vein, to a simple repetitive "sucking" behavior. This example shows the utility of variable length motif discovery. An entomologist using a classic motif search, say at the length of *12* seconds, might have believed that this insect *only* engaged in xylem ingestion during this time period. They would not have realized that the insect had to reposition itself at least twice.

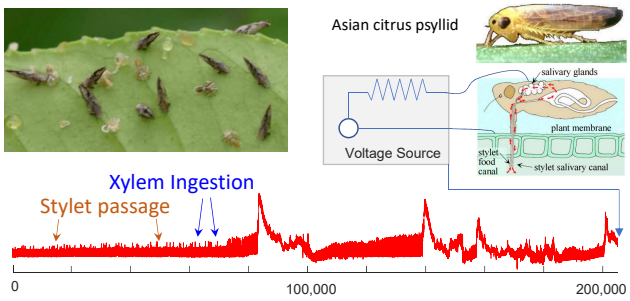

Figure 16: (*top.left*) A cluster of Asian citrus psyllids feeding on the leaf of a Valencia orange tree. (*top.right*) A psyllid was connected to an EPG apparatus, and its behavior was monitored for 5.5 hours. The identified motifs are depicted in Figure 1.

10 COMPLEXITY ANALYSIS

10.1 Motif Pairs Discovery

ComputeMatrixProfile. In line 15 of Algorithm 2, the time cost to compute a single distance profile is $O(n)$, where n is the number of subsequences of length ℓ. Therefore computing the n distance profiles takes $O(n^2)$ time. In line 18, computing the lower bounds of the smallest p entries of each distance profile takes $O(nlogp)$ additional time. The overall time complexity of the $ComputeMatrixProfile$ routine is thus $O(n^2logp)$. This routine is called at least once, for the first subsequence length of the range, namely $\ell = \ell_{min}$. In the worst case, it is executed for each length in the range (though, this never occurred in our experiments).

ComputeSubMP. In the best case, $ComputeSubMP$ can find the motif pair in $O(np)$ time, where n, is the total number of distance profiles. This means that, no distance profile computation takes place, since the condition in line 26 of Algorithm5 is satisfied. Otherwise, if we need to iterate over the non-valid distances profiles for finding the answer (which occurs rarely in practice), the time complexity reaches its worst case, $O(nClogn)$, with $C = n/p$. This is asymptotically faster than re-executing $ComputeMatrixProfile$, which takes $O(n^2logp)$ time. Note that, each non-valid distance profile (starting in line 30) is computed by using the primitives introduced in the *ComputeMatrixProfile* algorithm, only if its maximum lower bound is less than the smallest true distance *minDistABS*. This indicates that the distance profile for length *newL* may contain not yet computed distances smaller than *minDistABS*, which is our *best-so-far*. Therefore, the overall complexity of VALMOD is $O(n^2logp + (\ell_{max} - \ell_{min})np)$ in the best case, whereas the worst case time complexity is $O((\ell_{max} - \ell_{min})n^2logp)$. Clearly, the n^2logp factor dominates, since $(\ell_{max} - \ell_{min})$ acts as a constant. Nevertheless, the length range is not negligible, w.r.t the time performance, when we need to run a quadratic routine over it. If the worst case occurs often, then the performance will degenerate. However, this is not the case, as we show in the experimental evaluation.

10.2 Motif Sets Discovery

UpdateVALMPForMotifSets. The complexity of the $updateVALMPForMotifSets$ algorithm is $O(nlogK)$, where n is the length of the $VALMP$ structure, which is linearly scanned and updated. $O(logK)$ time is needed to retain the K best pairs of $VALMP$, using the heap structure in line 8.

ComputeVarLengthMotifSets. The final algorithm $computeVarLengthMotifSets$ takes $O(Kplogp)$ time, in the best case. This occurs, when after iterating the K pairs in $heapBestKPairs$, each partial distance profile of length p, contains all the elements in the range r. In this case, we just need an extra $O(plogp)$ time to sort its elements (line 7 and 14). On the other hand, the worst case time is bounded by $O(Knlogn)$, where n is the length of the input data series T. In this case, the algorithm needs to recompute K times the entire distance profile (line 11 and 18), at a unit cost of $O(nlogn)$ time.